Learning Bayesian Networks with the bnlearn R Package

Marco Scutari

University of Padova

Abstract

bnlearn is an R package (R Development Core Team 2010) which includes several algorithms for learning the structure of Bayesian networks with either discrete or continuous variables. Both constraint-based and score-based algorithms are implemented, and can use the functionality provided by the **snow** package (Tierney *et al.* 2008) to improve their performance via parallel computing. Several network scores and conditional independence algorithms are available for both the learning algorithms and independent use. Advanced plotting options are provided by the **Rgraphviz** package (Gentry *et al.* 2010).

Keywords: bayesian networks, R, structure learning algorithms, constraint-based algorithms, score-based algorithms, conditional independence tests.

1. Introduction

In recent years Bayesian networks have been used in many fields, from On-line Analytical Processing (OLAP) performance enhancement (Margaritis 2003) to medical service performance analysis (Acid *et al.* 2004), gene expression analysis (Friedman *et al.* 2000), breast cancer prognosis and epidemiology (Holmes and Jain 2008).

The high dimensionality of the data sets common in these domains have led to the development of several learning algorithms focused on reducing computational complexity while still learning the correct network. Some examples are the *grow-shrink* algorithm in Margaritis (2003), the *incremental association* algorithm and its derivatives in Tsamardinos *et al.* (2003) and in Yaramakala and Margaritis (2005), the *sparse candidate* algorithm in Friedman *et al.* (1999), the optimal reinsertion in Moore and Wong (2003) and the greedy equivalent search in Chickering (2002).

The aim of the **bnlearn** package is to provide a free implementation of some of these structure learning algorithms along with the conditional independence tests and network scores

used to construct the Bayesian network. Both discrete and continuous data are supported. Furthermore the learning algorithms can be chosen separately from the statistical criterion they are based on (which is usually not possible in the reference implementation provided by the algorithms' authors), so that the best combination for the data at hand can be used. The package is available from the Comprehensive R Archive Network (CRAN) at http://CRAN.R-project.org/package=bnlearn.

2. Bayesian networks

Bayesian networks are graphical models where nodes represent random variables (the two terms are used interchangeably in this article) and arrows represent probabilistic dependencies between them (Korb and Nicholson 2004).

The graphical structure $\mathcal{G} = (\mathbf{V}, A)$ of a Bayesian network is a *directed acyclic graph* (DAG), where $\mathbf{V}$ is the *node* (or *vertex*) *set* and A is the *arc* (or *edge*) *set*. The DAG defines a factorization of the joint probability distribution of $\mathbf{V} = \{X_1, X_2, \ldots, X_v\}$, often called the *global probability distribution*, into a set of *local probability distributions*, one for each variable. The form of the factorization is given by the *Markov property* of Bayesian networks (Korb and Nicholson 2004, Section 2.2.4), which states that every random variable X_i directly depends only on its parents Π_{X_i}:

$$P(X_1, \ldots, X_v) = \prod_{i=1}^{v} P(X_i \,|\, \Pi_{X_i}) \qquad \text{(for discrete variables)} \qquad (1)$$

$$f(X_1, \ldots, X_v) = \prod_{i=1}^{v} f(X_i \,|\, \Pi_{X_i}) \qquad \text{(for continuous variables).} \qquad (2)$$

The correspondence between conditional independence (of the random variables) and graphical separation (of the corresponding nodes of the graph) has been extended to an arbitrary triplet of disjoint subsets of $\mathbf{V}$ by Pearl (1988) with the *d-separation* (from *direction-dependent separation*). Therefore model selection algorithms first try to learn the graphical structure of the Bayesian network (hence the name of *structure learning algorithms*) and then estimate the parameters of the local distribution functions conditional on the learned structure. This two-step approach has the advantage that it considers one local distribution function at a time, and it does not require to model the global distribution function explicitly. Another advantage is that learning algorithms are able to scale to fit high-dimensional models without incurring in the so-called *curse of dimensionality*.

Although there are many possible choices for both the global and the local distribution functions, literature have focused mostly on two cases:

- *Multinomial data* (the *discrete case*): Both the global and the local distributions are multinomial, and are represented as probability or contingency tables. This is by far the most common assumption, and the corresponding Bayesian networks are usually referred to as *discrete Bayesian networks* (or simply as *Bayesian networks*).

- *Multivariate normal data* (the *continuous case*): The global distribution is multivariate normal, and the local distributions are normal random variables linked by linear constraints. These Bayesian networks are called *Gaussian Bayesian networks* in Geiger and Heckerman (1994), Neapolitan (2003) and most recent literature on the subject.

Other distributional assumptions lead to more complex learning algorithms (such as the non-parametric approach proposed by Bach and Jordan 2003) or present various limitations due to the difficulty of specifying the distribution functions in closed form (such as the approach to learn Bayesian network with mixed variables by Boettcher and Dethlefsen 2003, which does not allow a node associated with a continuous variable to be the parent of a node associated with a discrete variable).

3. Structure learning algorithms

Bayesian network structure learning algorithms can be grouped in two categories:

- *Constraint-based algorithms*: These algorithms learn the network structure by analyzing the probabilistic relations entailed by the Markov property of Bayesian networks with conditional independence tests and then constructing a graph which satisfies the corresponding d-separation statements. The resulting models are often interpreted as *causal models* even when learned from observational data (Pearl 1988).

- *Score-based algorithms*: These algorithms assign a score to each candidate Bayesian network and try to maximize it with some heuristic search algorithm. Greedy search algorithms (such as *hill-climbing* or *tabu search*) are a common choice, but almost any kind of search procedure can be used.

Constraint-based algorithms are all based on the *inductive causation* (IC) algorithm by Verma and Pearl (1991), which provides a theoretical framework for learning the structure causal models. It can be summarized in three steps:

1. First the *skeleton* of the network (the undirected graph underlying the network structure) is learned. Since an exhaustive search is computationally unfeasible for all but the most simple data sets, all learning algorithms use some kind of optimization such as restricting the search to the *Markov blanket* of each node (which includes the parents, the children and all the nodes that share a child with that particular node).

2. Set all direction of the arcs that are part of a *v-structure* (a triplet of nodes incident on a converging connection $X_j \rightarrow X_i \leftarrow X_k$).

3. Set the directions of the other arcs as needed to satisfy the acyclicity constraint.

Score-based algorithms on the other hand are simply applications of various general purpose heuristic search algorithms, such as *hill-climbing*, *tabu search*, *simulated annealing* and various *genetic algorithms*. The score function is usually *score-equivalent* (Chickering 1995), so that networks that define the same probability distribution are assigned the same score.

4. Package implementation

4.1. Structure learning algorithms

bnlearn implements the following constraint-based learning algorithms (the respective function names are reported in parenthesis):

- *Grow-shrink* (`gs`): Based on the *grow-shrink Markov blanket*, the simplest Markov blanket detection algorithm (Margaritis 2003) used in a structure learning algorithm.

- *Incremental association* (`iamb`): Based on the incremental association Markov blanket (IAMB) algorithm (Tsamardinos *et al.* 2003), which is based on a two-phase selection scheme (a forward selection followed by an attempt to remove false positives).

- *Fast incremental association* (`fast.iamb`): A variant of IAMB which uses speculative stepwise forward selection to reduce the number of conditional independence tests (Yaramakala and Margaritis 2005).

- *Interleaved incremental association* (`inter.iamb`): Another variant of IAMB which uses forward stepwise selection (Tsamardinos *et al.* 2003) to avoid false positives in the Markov blanket detection phase.

- *Max-min parents and children* (`mmpc`): A forward selection technique for neighbourhood detection based on the maximization of the minimum association measure observed with any subset of the nodes selected in the previous iterations (Tsamardinos *et al.* 2006). It learns the underlying structure of the Bayesian network (all the arcs are undirected, no attempt is made to detect their orientation).

Three implementations are provided for each algorithm:

- An optimized implementation (used by default) which uses backtracking to roughly halve the number of independence tests.

- An unoptimized implementation (used when the `optimized` argument is set to `FALSE`) which is faithful to the original description of the algorithm. This implementation is particularly useful for comparing the behaviour of different combinations of learning algorithms and statistical tests.

- A parallel implementation. It requires a running cluster set up with the `makeCluster` function from the **snow** package (Tierney *et al.* 2008, see also Schmidberger *et al.* 2009), which is passed to the function via the `cluster` argument.

The only available score-based learning algorithm is a *Hill-Climbing* (`hc`) greedy search on the space of directed graphs. The optimized implementation (again used by default) uses score caching, score decomposability and score equivalence to reduce the number of duplicated tests (Daly and Shen 2007). Random restarts, a configurable number of perturbing operations and a preseeded initial network structure can be used to avoid poor local maxima (with the `restart`, `perturb` and `start` arguments, respectively).

4.2. Conditional independence tests

Several conditional independence tests from information theory and classical statistics are available for use in constraint-based learning algorithms and the `ci.test` function. In both cases the test to be used is specified with the `test` argument (the label associated with each test is reported in parenthesis).

Conditional independence tests for discrete data are functions of the conditional probability tables implied by the graphical structure of the network through the observed frequencies

$\{n_{ijk}, i = 1, \ldots, R, j = 1, \ldots, C, k = 1, \ldots, L\}$ for the random variables X and Y and all the configurations of the conditioning variables $\mathbf{Z}$:

- *Mutual information*: An information-theoretic distance measure (Kullback 1959), defined as

$$\mathrm{MI}(X, Y \mid \mathbf{Z}) = \sum_{i=1}^{R} \sum_{j=1}^{C} \sum_{k=1}^{L} \frac{n_{ijk}}{n} \log \frac{n_{ijk} n_{++k}}{n_{i+k} n_{+jk}}. \qquad (3)$$

  It is proportional to the log-likelihood ratio test G^2 (they differ by a $2n$ factor, where n is the sample size) and it is related to the deviance of the tested models. Both the asymptotic χ^2 test (`mi`) and the Monte Carlo permutation test (`mc-mi`) described in Good (2005) are available.

- *Pearson's X^2*: The classical Pearson's X^2 test for contingency tables,

$$X^2(X, Y \mid \mathbf{Z}) = \sum_{i=1}^{R} \sum_{j=1}^{C} \sum_{k=1}^{L} \frac{(n_{ijk} - m_{ijk})^2}{m_{ijk}}, \qquad m_{ijk} = \frac{n_{i+k} n_{+jk}}{n_{++k}} \qquad (4)$$

  Again both the asymptotic χ^2 test (`x2`) and a Monte Carlo permutation test (`mc-x2`) from Good (2005) are available.

- *Fast mutual information* (`fmi`): A variant of the mutual information which is set to zero when there aren't at least five data per parameter, which is the usual threshold for establishing the correctness of the asymptotic χ^2 distribution. This is the same heuristic defined for the Fast-IAMB algorithm in Yaramakala and Margaritis (2005).

- *Akaike information criterion* (`aict`): An experimental AIC-based independence test, computed comparing the mutual information and the expected information gain. It rejects the null hypothesis if

$$\mathrm{MI}(X, Y \mid \mathbf{Z}) \geqslant \frac{(R-1)(C-1)L}{n}, \qquad (5)$$

  which corresponds to an increase in the AIC score of the network.

In the continuous case conditional independence tests are functions of the partial correlation coefficients $\rho_{XY \mid \mathbf{Z}}$ of X and Y given $\mathbf{Z}$:

- *Linear correlation*: The linear correlation coefficient $\rho_{XY \mid \mathbf{Z}}$. Both the asymptotic Student's t test (`cor`) and the Monte Carlo permutation test (`mc-cor`) described in Legendre (2000) are available.

- *Fisher's Z*: A transformation of the linear correlation coefficient used by commercial software (such as **TETRAD**) and the **pcalg** package (Kalisch and Bühlmann 2007), which implements the PC constraint-based learning algorithm (Spirtes *et al.* 2001). It is defined as

$$Z(X, Y \mid \mathbf{Z}) = \frac{1}{2} \sqrt{n - |\mathbf{Z}| - 3} \log \frac{1 + \rho_{XY \mid \mathbf{Z}}}{1 - \rho_{XY \mid \mathbf{Z}}}. \qquad (6)$$

  Both the asymptotic normal test (`zf`) and the Monte Carlo permutation test (`mc-zf`) are available.

- *Mutual information* (`mi-g`): An information-theoretic distance measure (Kullback 1959), defined as

$$\text{MI}_{\text{g}}(X, Y \mid \mathbf{Z}) = -\frac{1}{2}\log(1 - \rho^2_{XY \mid \mathbf{z}}). \tag{7}$$

  It has the same relationship with the log-likelihood ratio as the corresponding test defined in the discrete case.

4.3. Network scores

Several score functions are available for use in the hill-climbing algorithm and the `score` function. The score to be used is specified with the `score` argument in `hc` and with the `type` argument in the `score` function (the label associated with each score is reported in parenthesis).

In the discrete case the following score functions are implemented:

- The *likelihood* (`lik`) and *log-likelihood* (`loglik`) scores, which are equivalent to the *entropy measure* used by **Weka** (Witten and Frank 2005).

- The *Akaike* (`aic`) and *Bayesian* (`bic`) *information criterion* scores, defined as

$$\text{AIC} = \log \text{L}(X_1, \ldots, X_v) - d \qquad \text{BIC} = \log \text{L}(X_1, \ldots, X_v) - \frac{d}{2}\log n \tag{8}$$

  The latter is equivalent to the *minimum description length* described by Rissanen (1978) and used as a Bayesian network score in Lam and Bacchus (1994).

- The logarithm of the *Bayesian Dirichlet equivalent* score (`bde`), a score equivalent Dirichlet posterior density (Heckerman *et al.* 1995).

- The logarithm of the *K2* score (`k2`), another Dirichlet posterior density (Cooper and Herskovits 1992) defined as

$$\text{K2} = \prod_{i=1}^{v} \text{K2}(X_i), \qquad \text{K2}(X_i) = \prod_{j=1}^{L_i} \frac{(R_i - 1)!}{\left(\sum_{k=1}^{R_i} n_{ijk} + R_i - 1\right)!} \prod_{k=1}^{R_i} n_{ijk}! \tag{9}$$

  and originally used in the structure learning algorithm of the same name. Unlike the `bde` score `k2` is not score equivalent.

The only score available for the continuous case is a score equivalent Gaussian posterior density (`bge`), which follows a Wishart distribution (Geiger and Heckerman 1994).

4.4. Arc whitelisting and blacklisting

Prior information on the data, such as the ones elicited from experts in the relevant fields, can be integrated in all learning algorithms by means of the `blacklist` and `whitelist` arguments. Both of them accept a set of arcs which is guaranteed to be either present (for the former) or missing (for the latter) from the Bayesian network; any arc whitelisted and blacklisted at the same time is assumed to be whitelisted, and is thus removed from the blacklist.

This combination represents a very flexible way to describe any arbitrary set of assumptions on the data, and is also able to deal with partially directed graphs:

- Any arc whitelisted in both directions (i.e., both $A \to B$ and $B \to A$ are whitelisted) is present in the graph, but the choice of its direction is left to the learning algorithm. Therefore one of $A \to B$, $B \to A$ and $A - B$ is guaranteed to be in the Bayesian network.

- Any arc blacklisted in both directions, as well as the corresponding undirected arc, is never present in the graph. Therefore if both $A \to B$ and $B \to A$ are blacklisted, also $A - B$ is considered blacklisted.

- Any arc whitelisted in one of its possible directions (i.e., $A \to B$ is whitelisted, but $B \to A$ is not) is guaranteed to be present in the graph in the specified direction. This effectively amounts to blacklisting both the corresponding undirected arc $(A - B)$ and its reverse $(B \to A)$.

- Any arc blacklisted in one of its possible directions (i.e., $A \to B$ is blacklisted, but $B \to A$ is not) is never present in the graph. The same holds for $A - B$, but not for $B \to A$.

5. A simple example

In this section **bnlearn** will be used to analyze a small data set, `learning.test`. It's included in the package itself along with other real word and synthetic data sets, and is used in the example sections throughout the manual pages due to its simple structure.

5.1. Loading the package

bnlearn and its dependencies (the **utils** package, which is bundled with R) are available from CRAN, as are the suggested packages **snow** and **graph** (Gentleman *et al.* 2010). The other suggested package, **Rgraphviz** (Gentry *et al.* 2010), can be installed from **Bioconductor** and is loaded along with **bnlearn** if present.

```
R> library("bnlearn")
Loading required package: Rgraphviz
Loading required package: graph
Loading required package: grid
Package Rgraphviz loaded successfully.
```

5.2. Learning a Bayesian network from data

Once **bnlearn** is loaded, `learning.test` itself can be loaded into a data frame of the same name with a call to `data`.

```
R> data(learning.test)
R> str(learning.test)
'data.frame':       5000 obs. of  6 variables:
 $ A: Factor w/ 3 levels "a","b","c": 2 2 1 1 1 3 3 2 2 2 ...
 $ B: Factor w/ 3 levels "a","b","c": 3 1 1 1 1 3 3 2 2 1 ...
 $ C: Factor w/ 3 levels "a","b","c": 2 3 1 1 2 1 2 1 2 2 ...
```

```
$ D: Factor w/ 3 levels "a","b","c": 1 1 1 1 3 3 3 2 1 1 ...
$ E: Factor w/ 3 levels "a","b","c": 2 2 1 2 1 3 3 2 3 1 ...
$ F: Factor w/ 2 levels "a","b": 2 2 1 2 1 1 1 2 1 1 ...
```

`learning.test` contains six discrete variables, stored as factors, each with 2 (for F) or 3 (for A, B, C, D and E) levels. The structure of the Bayesian network associated with this data set can be learned for example with the grow-shrink algorithm, implemented in the `gs` function, and stored in an object of class `bn`.

```
R> bn.gs <- gs(learning.test)
R> bn.gs

  Bayesian network learned via Constraint-based methods

  model:
    [partially directed graph]
  nodes:                              6
  arcs:                               5
    undirected arcs:                  1
    directed arcs:                    4
  average markov blanket size:        2.33
  average neighbourhood size:         1.67
  average branching factor:           0.67

  learning algorithm:                 Grow-Shrink
  conditional independence test:      Mutual Information (discrete)
  alpha threshold:                    0.05
  tests used in the learning procedure: 43
  optimized:                          TRUE
```

Other constraint-based algorithms return the same partially directed network structure (again as an object of class `bn`), as can be readily seen with `compare`.

```
R> bn2 <- iamb(learning.test)
R> bn3 <- fast.iamb(learning.test)
R> bn4 <- inter.iamb(learning.test)

R> compare(bn.gs, bn2)
[1] TRUE
R> compare(bn.gs, bn3)
[1] TRUE
R> compare(bn.gs, bn4)
[1] TRUE
```

On the other hand hill-climbing results in a completely directed network, which differs from the previous one because the arc between A and B is directed ($A \rightarrow B$ instead of $A - B$).

```
R> bn.hc <- hc(learning.test, score = "aic")
R> bn.hc

  Bayesian network learned via Score-based methods

  model:
    [A][C][F][B|A][D|A:C][E|B:F]
  nodes:                                 6
  arcs:                                  5
    undirected arcs:                     0
    directed arcs:                       5
  average markov blanket size:           2.33
  average neighbourhood size:            1.67
  average branching factor:              0.83

  learning algorithm:                    Hill-Climbing
  score:                                 Akaike Information Criterion
  penalization coefficient:              1
  tests used in the learning procedure:  40
  optimized:                             TRUE

R> compare(bn.hc, bn.gs)
[1] FALSE
```

Another way to compare the two network structures is to plot them side by side and highlight the differing arcs. This can be done either with the `plot` function (see Figure 1):

```
R> par(mfrow = c(1, 2))
R> plot(bn.gs, main = "Constraint-based algorithms", highlight = c("A", "B"))
R> plot(bn.hc, main = "Hill-Climbing", highlight = c("A", "B"))
```

or with the more versatile `graphviz.plot`:

```
R> par(mfrow = c(1, 2))
R> highlight.opts <- list(nodes = c("A", "B"), arcs = c("A", "B"),
+    col = "red", fill = "grey")
R> graphviz.plot(bn.hc, highlight = highlight.opts)
R> graphviz.plot(bn.gs, highlight = highlight.opts)
```

which produces a better output for large graphs thanks to the functionality provided by the **Rgraphviz** package.

The network structure learned by `gs`, `iamb`, `fast.iamb` and `inter.iamb` is equivalent to the one learned by `hc`; changing the arc $A - B$ to either $A \rightarrow B$ or to $B \rightarrow A$ results in networks with the same score because of the score equivalence property (which holds for all the implemented score functions with the exception of $K2$). Therefore if there is any prior information about the relationship between A and B the appropriate direction can be whitelisted (or its reverse can be blacklisted, which is equivalent in this case).

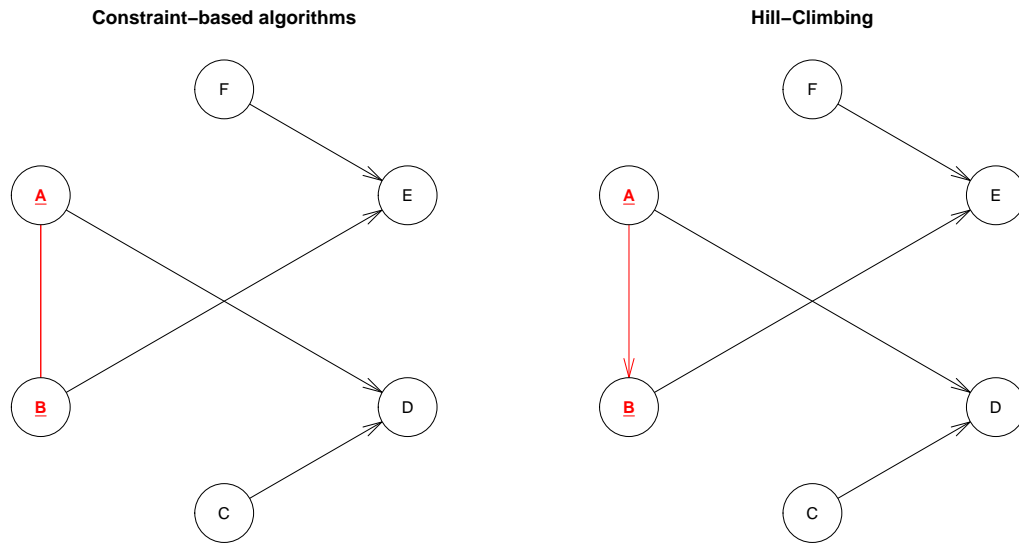**Constraint–based algorithms**   **Hill–Climbing**

Figure 1: side by side comparison of the Bayesian network structures learned by constraint-based (`gs`, `iamb`, `fast.iamb` and `inter.iamb`) and score-based (`hc`) algorithms. Arcs which differ between the two network structures are plotted in red.

```
R> bn.AB <- gs(learning.test, blacklist = c("B", "A"))
R> compare(bn.AB, bn.hc)
[1] TRUE
R> score(bn.AB, learning.test, type = "bde")
[1] -24002.36
R> bn.BA <- gs(learning.test, blacklist = c("A", "B"))
R> score(bn.BA, learning.test, type = "bde")
[1] -24002.36
```

5.3. Network analysis and manipulation

The structure of a Bayesian network is uniquely specified if its graph is completely directed. In that case it can be represented as a string with the `modelstring` function

```
R> modelstring(bn.hc)
[1] "[A][C][F][B|A][D|A:C][E|B:F]"
```

whose output is also included in the `print` method for the objects of class `bn`. Each node is printed in square brackets along with all its parents (which are reported after a pipe as a colon-separated list), and its position in the string depends on the partial ordering defined by the network structure. The same syntax is used in **deal** (Boettcher and Dethlefsen 2003), an R package for learning Bayesian networks from mixed data.

Partially directed graphs can be transformed into completely directed ones with the `set.arc`, `drop.arc` and `reverse.arc` functions. For example the direction of the arc $A - B$ in the

`bn.gs` object can be set to $A \rightarrow B$, so that the resulting network structure is identical to the one learned by the hill-climbing algorithm.

```
R> undirected.arcs(bn.gs)
     from to
[1,] "A"  "B"
[2,] "B"  "A"
R> bn.dag <- set.arc(bn.gs, "A", "B")
R> modelstring(bn.dag)
[1] "[A][C][F][B|A][D|A:C][E|B:F]"
R> compare(bn.dag, bn.hc)
[1] TRUE
```

Acyclicity is always preserved, as these commands return an error if the requested changes would result in a cyclic graph.

```
R> set.arc(bn.hc, "E", "A")
Error in arc.operations(x = x, from = from, to = to, op = "set",
check.cycles = check.cycles,  :
  the resulting graph contains cycles.
```

Further information on the network structure can be extracted from any `bn` object with the following functions:

- whether the network structure is acyclic (`acyclic`) or completely directed (`directed`);

- the labels of the nodes (`nodes`), of the root nodes (`root.nodes`) and of the leaf nodes (`leaf.nodes`);

- the directed arcs (`directed.arcs`) of the network, the undirected ones (`undirected.arcs`) or both of them (`arcs`);

- the adjacency matrix (`amat`) and the number of parameters (`nparams`) associated with the network structure;

- the parents (`parents`), children (`children`), Markov blanket (`mb`), and neighbourhood (`nbr`) of each node.

The `arcs`, `amat` and `modelstring` functions can also be used in combination with `empty.graph` to create a `bn` object with a specific structure from scratch:

```
R> other <- empty.graph(nodes = nodes(bn.hc))

R> arcs(other) <- data.frame(
+    from = c("A", "A", "B", "D"),
+    to = c("E", "F", "C", "E"))
R> other

  Randomly generated Bayesian network
```

```
model:
  [A][B][D][C|B][E|A:D][F|A]
nodes:                            6
arcs:                            4
  undirected arcs:               0
  directed arcs:                 4
average markov blanket size:     1.67
average neighbourhood size:      1.33
average branching factor:        0.67

generation algorithm:            Empty
```

This is particularly useful to compare different network structures for the same data, for example to verify the goodness of fit of the learned network with respect to a particular score function.

```
R> score(other, data = learning.test, type = "aic")
[1] -28019.79
R> score(bn.hc, data = learning.test, type = "aic")
[1] -23873.13
```

5.4. Debugging utilities and diagnostics

Many of the functions of the **bnlearn** package are able to print additional diagnostic messages if called with the `debug` argument set to `TRUE`. This is especially useful to study the behaviour of the learning algorithms in specific settings and to investigate anomalies in their results (which may be due to an insufficient sample size for the asymptotic distribution of the tests to be valid, for example). For example the debugging output of the call to `gs` previously used to produce the `bn.gs` object reports the exact sequence of conditional independence tests performed by the learning algorithm, along with the effects of the backtracking optimizations (some parts are omitted for brevity).

```
R> gs(learning.test, debug = TRUE)
----------------------------------------------------------------
* learning markov blanket of A .
  * checking node B for inclusion.
    > node B included in the markov blanket ( p-value: 0 ).
    > markov blanket now is ' B '.
  * checking node C for inclusion.
    > A indep. C given ' B ' ( p-value: 0.8743202 ).
  * checking node D for inclusion.
    > node D included in the markov blanket ( p-value: 0 ).
    > markov blanket now is ' B D '.
  * checking node E for inclusion.
    > A indep. E given ' B D ' ( p-value: 0.5193303 ).
```

```
   * checking node F for inclusion.
     > A indep. F given ' B D ' ( p-value: 0.07368042 ).
   * checking node C for inclusion.
     > node C included in the markov blanket ( p-value: 1.023194e-254 ).
     > markov blanket now is ' B D C '.
   * checking node E for inclusion.
     > A indep. E given ' B D C ' ( p-value: 0.5091863 ).
   * checking node F for inclusion.
     > A indep. F given ' B D C ' ( p-value: 0.3318902 ).
   * checking node B for exclusion (shrinking phase).
     > node B remains in the markov blanket. ( p-value: 9.224694e-291 )
   * checking node D for exclusion (shrinking phase).
     > node D remains in the markov blanket. ( p-value: 0 )
----------------------------------------------------------------
* learning markov blanket of B .
[...]
----------------------------------------------------------------
* learning markov blanket of F .
     * known good (backtracking): ' B E '.
     * known bad (backtracking): ' A C D '.
     * nodes still to be tested for inclusion: ' '.
----------------------------------------------------------------
* checking consistency of markov blankets.
[...]
----------------------------------------------------------------
* learning neighbourhood of A .
  * blacklisted nodes: ' '
  * whitelisted nodes: ' '
  * starting with neighbourhood: ' B D C '
  * checking node B for neighbourhood.
    > dsep.set = ' E F '
    > trying conditioning subset ' '.
    > node B is still a neighbour of A . ( p-value: 0 )
    > trying conditioning subset ' E '.
    > node B is still a neighbour of A . ( p-value: 0 )
    > trying conditioning subset ' F '.
    > node B is still a neighbour of A . ( p-value: 0 )
    > trying conditioning subset ' E F '.
    > node B is still a neighbour of A . ( p-value: 0 )
  * checking node D for neighbourhood.
    > dsep.set = ' C '
    > trying conditioning subset ' '.
    > node D is still a neighbour of A . ( p-value: 0 )
    > trying conditioning subset ' C '.
    > node D is still a neighbour of A . ( p-value: 0 )
  * checking node C for neighbourhood.
    > dsep.set = ' D '
```

```
    > trying conditioning subset ' '.
    > node C is not a neighbour of A . ( p-value: 0.8598334 )
-----------------------------------------------------------------
* learning neighbourhood of B .
[...]
-----------------------------------------------------------------
* learning neighbourhood of F .
  * blacklisted nodes: ' '
  * whitelisted nodes: ' '
  * starting with neighbourhood: ' E '
  * known good (backtracking): ' E '.
  * known bad (backtracking): ' A B C D '.
-----------------------------------------------------------------
* checking consistency of neighbourhood sets.
[...]
-----------------------------------------------------------------
* v-structures centered on D .
  * checking A -> D <- C
    > chosen d-separating set: ' '
    > testing A vs C given  D ( 0 )
    @ detected v-structure A -> D <- C
-----------------------------------------------------------------
* v-structures centered on E .
  * checking B -> E <- F
    > chosen d-separating set: ' '
    > testing B vs F given  E ( 1.354269e-50 )
    @ detected v-structure B -> E <- F
-----------------------------------------------------------------
* v-structures centered on F .
-----------------------------------------------------------------
* applying v-structure A -> D <- C ( 0.000000e+00 )
* applying v-structure B -> E <- F ( 1.354269e-50 )
-----------------------------------------------------------------
* detecting cycles ...
[...]
-----------------------------------------------------------------
* propagating directions for the following undirected arcs:
[...]
```

Other functions which provide useful diagnostics include (but are not limited to) `compare` (which reports the differences between the two network structures with respect to arcs, parents and children for each node), `score` and `nparams` (which report the number of parameters and the contribution of each node to the network score, respectively).

6. Practical examples

6.1. The ALARM network

The ALARM ("A Logical Alarm Reduction Mechanism") network from Beinlich *et al.* (1989) is a Bayesian network designed to provide an alarm message system for patient monitoring. It has been widely used (see for example Tsamardinos *et al.* 2006 and Friedman *et al.* 1999) as a case study to evaluate the performance of new structure learning algorithms.
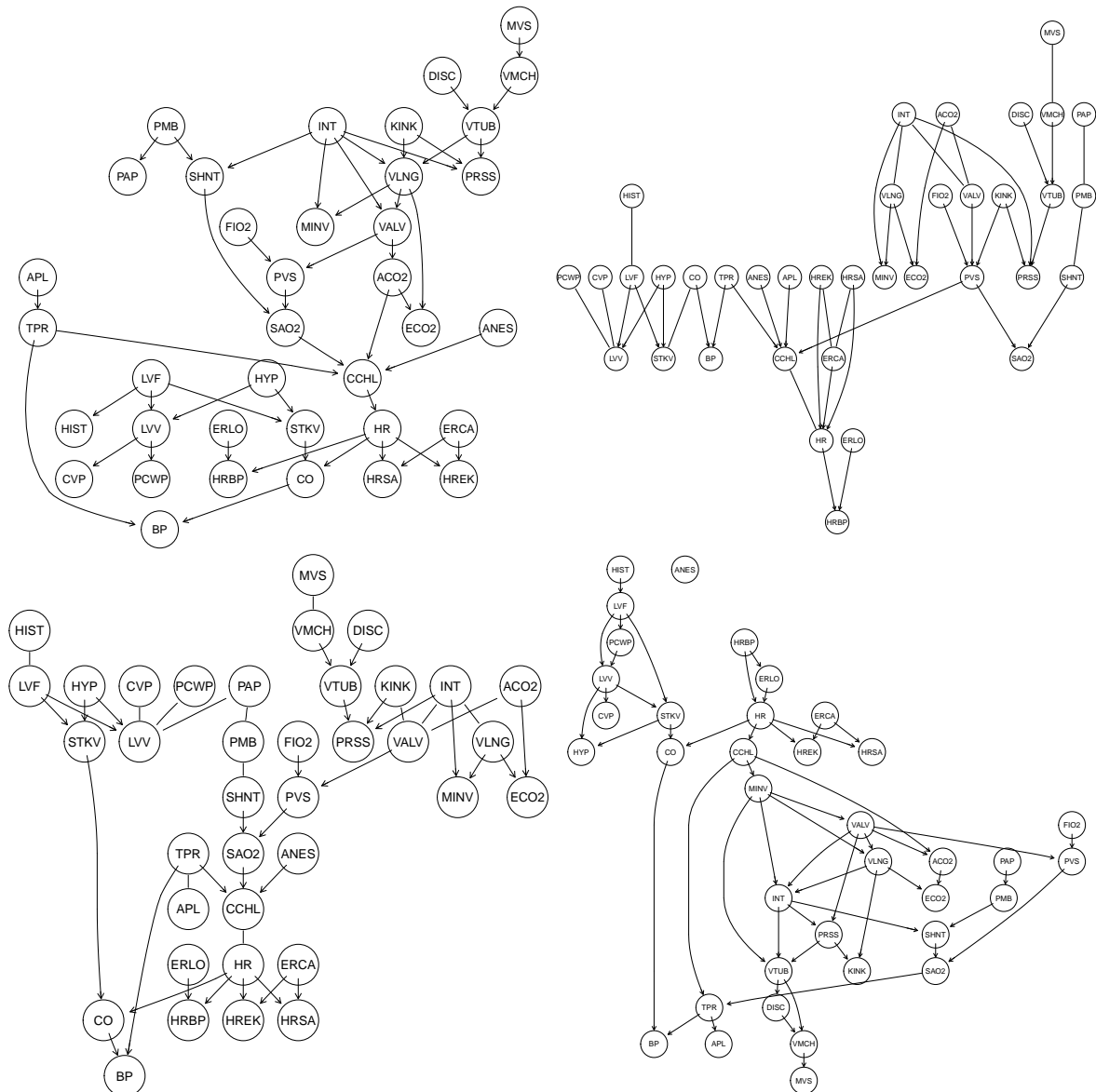

Figure 2: The ALARM data set: the original network structure (top left) and the network structures learned by `gs` (top right), `inter.iamb` (bottom left) and `hc` (bottom right).

	gs	iamb	fast.iamb	inter.iamb	hc
independence tests / network comparisons	1727	2874	2398	3106	2841
learned arcs (directed/undirected)	42 (29/13)	43 (29/14)	45 (32/13)	43 (30/13)	53 (53/0)
execution time	13.54360	17.63735	14.80890	18.59825	72.38705

Table 1: Performance of implemented learning algorithms with the `alarm` data set, measured in the number of conditional independence tests (for constraint-based algorithms) or network score comparisons (for score-based algorithms), the number of arcs and the execution time on an Intel Core 2 Duo machine with 1GB of RAM.

The `alarm` data set includes a sample of size 20000 generated from this network, which contains 37 discrete variables (with two to four levels each) and 46 arcs. Every learning algorithm implemented in **bnlearn** (except `mmpc`) is capable of recovering the ALARM network to within a few arcs and arc directions (see Figure 2).

```
R> alarm.gs <- gs(alarm)
R> alarm.iamb <- iamb(alarm)
R> alarm.fast.iamb <- fast.iamb(alarm)
R> alarm.inter.iamb <- inter.iamb(alarm)
R> alarm.mmpc <- mmpc(alarm)
R> alarm.hc <- hc(alarm, score = "bic")
```

The number of conditional independence tests, which provides an implementation independent performance indicator, is similar in all constraint-based algorithms (see Table 1); the same holds for the number of network score comparisons performed by `hc`, even though the learned network has about ten more arcs than the other ones.

The quality of the learned network improves significantly if the Monte Carlo versions of the tests are used instead of the parametric ones, as probability structure of the ALARM network results in many sparse contingency tables. For example a side by side comparison of the two versions of Pearson's X^2 test shows that the use of the nonparametric tests leads to the correct identification of all but five arcs, instead of the 12 missed with the parametric tests.

```
R> dag <- empty.graph(names(alarm))
R> modelstring(dag) <- paste("[HIST|LVF][CVP|LVV][PCWP|LVV][HYP][LVV|HYP:LVF]",
+   "[LVF][STKV|HYP:LVF][ERLO][HRBP|ERLO:HR][HREK|ERCA:HR][ERCA][HRSA|ERCA:HR]",
+   "[ANES][APL][TPR|APL][ECO2|ACO2:VLNG][KINK][MINV|INT:VLNG][FIO2]",
+   "[PVS|FIO2:VALV][SAO2|PVS:SHNT][PAP|PMB][PMB][SHNT|INT:PMB][INT]",
+   "[PRSS|INT:KINK:VTUB][DISC][MVS][VMCH|MVS][VTUB|DISC:VMCH]",
+   "[VLNG|INT:KINK:VTUB][VALV|INT:VLNG][ACO2|VALV][CCHL|ACO2:ANES:SAO2:TPR]",
+   "[HR|CCHL][CO|HR:STKV][BP|CO:TPR]", sep = "")
R> alarm.gs <- gs(alarm, test = "x2")
R> alarm.mc <- gs(alarm, test = "mc-x2", B = 10000)
R> par(mfrow = c(1,2), omi = rep(0, 4), mar = c(1, 0, 1, 0))
R> graphviz.plot(dag, highlight = list(arcs = arcs(alarm.gs)))
R> graphviz.plot(dag, highlight = list(arcs = arcs(alarm.mc)))
```
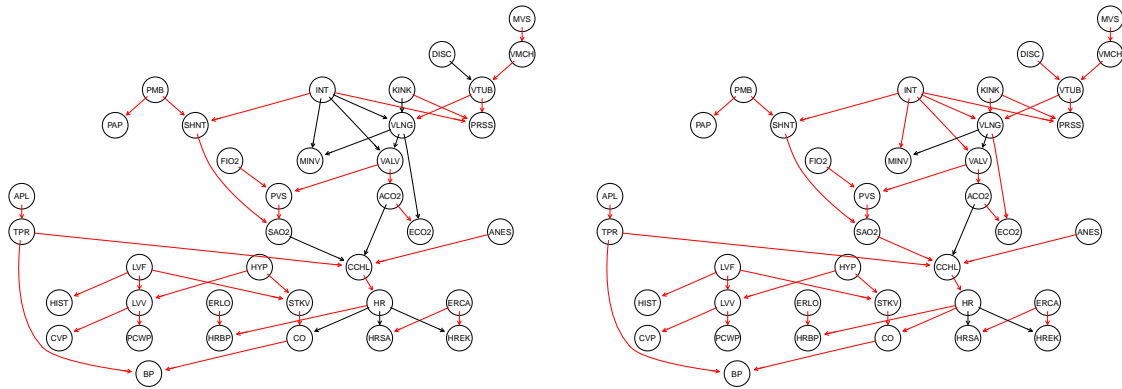
Figure 3: Side by side comparison of the network structures learned from the `alarm` data set by the grow-shrink algorithm with the parametric (on the left) and nonparametric (on the right) versions of Pearson's X^2 test. The arcs of the true network structure present in each case are highlighted in red.

6.2. The examination marks data set

The `marks` data set is a small data set studied in Mardia *et al.* (1979), Whittaker (1990) and Edwards (2000). It contains five continuous variables, the examination marks for 88 students in five different subjects (mechanics, vectors, algebra, analysis and statistics).

```
R> data(marks)
R> str(marks)
'data.frame':        88 obs. of  5 variables:
 $ MECH: num  77 63 75 55 63 53 51 59 62 64 ...
 $ VECT: num  82 78 73 72 63 61 67 70 60 72 ...
 $ ALG : num  67 80 71 63 65 72 65 68 58 60 ...
 $ ANL : num  67 70 66 70 70 64 65 62 62 62 ...
 $ STAT: num  81 81 81 68 63 73 68 56 70 45 ...
```

The purpose of the analysis was to find a suitable way to combine or average these marks. Since they are obviously correlated, the exact weights they are assigned depend on the estimated dependence structure of the data.

Under the assumption of multivariate normality this analysis requires the examination of the partial correlation coefficients, some of which are clearly not significative:

```
R> ci.test("MECH", "ANL", "ALG", data = marks)

        Pearson's Linear Correlation

data:  MECH ~ ANL | ALG
cor = 0.0352, df = 85, p-value = 0.7459
alternative hypothesis: true value is not equal to 0
```
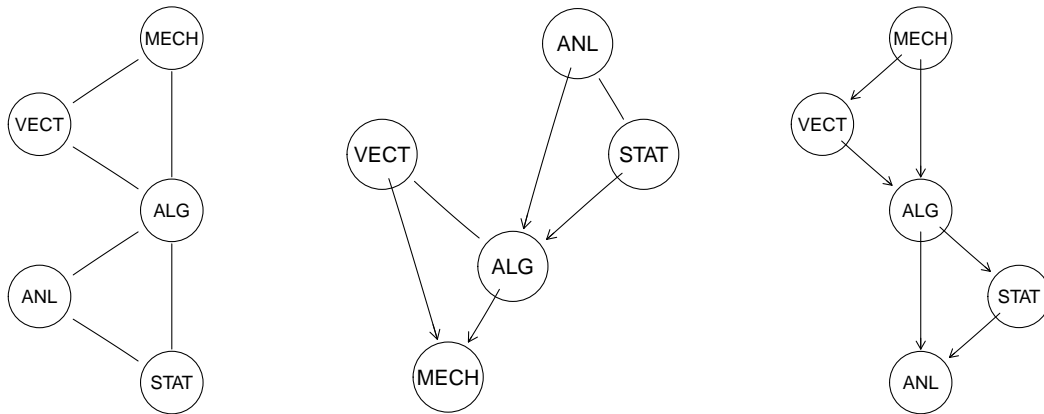
Figure 4: Graphical models learned from the marks data set. From left to right: the network learned by `mmpc` (which is identical to the Gaussian graphical model in Edwards 2000), the one learned by the other constraint-based algorithms and the one learned by `hc`.

```
R> ci.test("STAT", "VECT", "ALG", data = marks)

        Pearson's Linear Correlation

data:  STAT ~ VECT | ALG
cor = 0.0527, df = 85, p-value = 0.628
alternative hypothesis: true value is not equal to 0
```

This is confirmed by the other conditional independence tests, both parametric and nonparametric; for example:

```
R> ci.test("STAT", "VECT", "ALG", data = marks, test = "zf")$p.value
[1] 0.6289112
R> ci.test("STAT", "VECT", "ALG", data = marks, test = "mc-cor")$p.value
[1] 0.6332
R> ci.test("STAT", "VECT", "ALG", data = marks, test = "mi-g")$p.value
[1] 0.6209406
R> ci.test("STAT", "VECT", "ALG", data = marks, test = "mc-mi-g")$p.value
[1] 0.6226
```

All learning algorithms result in very similar network structures, which agree up to the direction of the arcs (see Figure 4). In all models the marks for analysis and statistics are conditionally independent from the ones for mechanics and vectors, given algebra. The structure of the graph suggests that the latter is essential in the overall evaluation of the examination.

7. Other packages for learning Bayesian networks

There exist other packages in R which are able to either learn the structure of a Bayesian network or fit and manipulate its parameters. Some examples are **pcalg** (Kalisch and Bühlmann

2007), which implements the PC algorithm and focuses on the causal interpretation of Bayesian networks; **deal** (Boettcher and Dethlefsen 2003), which implements a hill-climbing search for mixed data; and the suite composed by **gRbase** (Dethlefsen and Højsgaard 2005), **gRain** (Højsgaard 2010), **gRc** (Højsgaard and Lauritzen 2007), which implements various exact and approximate inference procedures.

However, none of these packages is as versatile as **bnlearn** for learning the structure of Bayesian networks. **deal** and **pcalg** implement a single learning algorithm, even though are able to handle both discrete and continuous data. Furthermore, the PC algorithm has a poor performance in terms of speed and accuracy compared to newer constraint-based algorithms such as grow-shrink and IAMB (Tsamardinos *et al.* 2003). **bnlearn** also offers a wider selection of network scores and conditional independence tests; in particular it's the only R package able to learn the structure of Bayesian networks using permutation tests, which are superior to the corresponding asymptotic tests at low sample sizes.

8. Conclusions

bnlearn is an R package which provides a free implementation of some Bayesian network structure learning algorithms appeared in recent literature, enhanced with algorithmic optimizations and support for parallel computing. Many score functions and conditional independence tests are provided for both independent use and the learning algorithms themselves.

bnlearn is designed to provide the versatility needed to handle experimental data analysis. It handles both discrete and continuous data, and it supports any combination of the implemented learning algorithms and either network scores (for score-based algorithms) or conditional independence tests (for constraints-based algorithms). Furthermore it simplifies the analysis of the learned networks by providing a single object class (`bn`) for all the algorithms and a set of utility functions to perform descriptive statistics and basic inference procedures.

Acknowledgments

References

Acid S, de Campos LM, Fernandez-Luna J, Rodriguez S, Rodriguez J, Salcedo J (2004). "A Comparison of Learning Algorithms for Bayesian Networks: A Case Study Based on Data from An Emergency Medical Service." *Artificial Intelligence in Medicine*, **30**, 215–232.

Bach FR, Jordan MI (2003). "Learning Graphical Models with Mercer Kernels." In *Advances in Neural Information Processing Systems (NIPS) 15*, pp. 1009–1016. MIT Press.

Beinlich IA, Suermondt HJ, Chavez RM, Cooper GF (1989). "The ALARM Monitoring System: A Case Study with Two Probabilistic Inference Techniques for Belief Networks." In *Proceedings of the 2nd European Conference on Artificial Intelligence in Medicine*, pp. 247–256. Springer-Verlag. URL http://www.cs.huji.ac.il/labs/compbio/Repository/Datasets/alarm/alarm.htm.

Boettcher SG, Dethlefsen C (2003). "**deal**: A Package for Learning Bayesian Networks." *Journal of Statistical Software*, **8**(20), 1–40. URL http://www.jstatsoft.org/v08/i20/.

Chickering DM (1995). "A Transformational Characterization of Equivalent Bayesian Network Structures." In *UAI '95: Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence*, pp. 87–98. Morgan Kaufmann.

Chickering DM (2002). "Optimal Structure Identification with Greedy Search." *Journal of Machine Learning Research*, **3**, 507–554.

Cooper GF, Herskovits E (1992). "A Bayesian Method for the Induction of Probabilistic Networks from Data." *Machine Learning*, **9**(4), 309–347.

Daly R, Shen Q (2007). "Methods to Accelerate the Learning of Bayesian Network Structures." In *Proceedings of the 2007 UK Workshop on Computational Intelligence*. Imperial College, London.

Dethlefsen C, Højsgaard S (2005). "A Common Platform for Graphical Models in R: The **gRbase** Package." *Journal of Statistical Software*, **14**(17), 1–12. URL http://www.jstatsoft.org/v14/i17/.

Edwards DI (2000). *Introduction to Graphical Modelling*. Springer-Verlag, New York.

Friedman N, Linial M, Nachman I (2000). "Using Bayesian Networks to Analyze Expression Data." *Journal of Computational Biology*, **7**, 601–620.

Friedman N, Pe'er D, Nachman I (1999). "Learning Bayesian Network Structure from Massive Datasets: The "Sparse Candidate" Algorithm." In *Proceedings of Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 206–221. Morgan Kaufmann.

Geiger D, Heckerman D (1994). "Learning Gaussian Networks." *Technical report*, Microsoft Research, Redmond, Washington. Available as Technical Report MSR-TR-94-10.

Gentleman R, Whalen E, Huber W, Falcon S (2010). **graph**: *A Package to Handle Graph Data Structures*. R package version 1.26.0, URL http://CRAN.R-project.org/.

Gentry J, Long L, Gentleman R, Falcon S, Hahne F, Sarkar D (2010). **Rgraphviz**: *Provides Plotting Capabilities for R Graph Objects*. R package version 1.26.0, URL http://www.bioconductor.org/packages/2.6/bioc/html/Rgraphviz.html.

Good P (2005). *Permutation, Parametric and Bootstrap Tests of Hypotheses*. 3rd edition. Springer-Verlag, New York.

Heckerman D, Geiger D, Chickering DM (1995). "Learning Bayesian Networks: The Combination of Knowledge and Statistical Data." *Machine Learning*, **20**(3), 197–243. Available as Technical Report MSR-TR-94-09.

Højsgaard S (2010). *gRain: Graphical Independence Networks*. R package version 0.8.5, URL http://CRAN.R-project.org/package=gRain.

Højsgaard S, Lauritzen SL (2007). "Inference in Graphical Gaussian Models with Edge and Vertex Symmetries with the **gRc** Package for R." *Journal of Statistical Software*, **23**(6), 1–26. URL http://www.jstatsoft.org/v23/i06/.

Holmes DE, Jain LC (eds.) (2008). *Innovations in Bayesian Networks: Theory and Applications*, volume 156 of *Studies in Computational Intelligence*. Springer-Verlag, New York.

Kalisch M, Bühlmann P (2007). "Estimating High-Dimensional Directed Acyclic Graphs with the PC-Algorithm." *Journal of Machine Learning Research*, **8**, 613–66.

Korb K, Nicholson A (2004). *Bayesian Artificial Intelligence*. Chapman and Hall.

Kullback S (1959). *Information Theory and Statistics*. John Wiley & Sons, Hoboken.

Lam W, Bacchus F (1994). "Learning Bayesian Belief Networks: An Approach Based on the MDL Principle." *Computational Intelligence*, **10**, 269–293.

Legendre P (2000). "Comparison of Permutation Methods for the Partial Correlation and Partial Mantel Tests." *Journal of Statistical Computation and Simulation*, **67**, 37–73.

Mardia KV, Kent JT, Bibby JM (1979). *Multivariate Analysis*. Academic Press.

Margaritis D (2003). *Learning Bayesian Network Model Structure from Data*. Ph.D. thesis, School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA. Available as Technical Report CMU-CS-03-153.

Moore A, Wong W (2003). "Optimal Reinsertion: A New Search Operator for Accelerated and More Accurate Bayesian Network Structure Learning." In *Proceedings of the 20th International Conference on Machine Learning (ICML '03)*, pp. 552–559. AAAI Press.

Neapolitan RE (2003). *Learning Bayesian Networks*. Prentice Hall.

Pearl J (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann.

R Development Core Team (2010). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

Rissanen J (1978). "Modeling by Shortest Data Description." *Automatica*, **14**, 465–471.

Schmidberger M, Morgan M, Eddelbuettel D, Yu H, Tierney L, Mansmann U (2009). "State of the Art in Parallel Computing with R." *Journal of Statistical Software*, **31**(1), 1–27. URL http://www.jstatsoft.org/v31/i01/.

Spirtes P, Glymour C, Scheines R (2001). *Causation, Prediction and Search*. MIT Press.

Tierney L, Rossini AJ, Li N, Sevcikova H (2008). *snow: Simple Network of Workstations*. R package version 0.3-3, URL http://CRAN.R-project.org/package=snow.

Tsamardinos I, Aliferis CF, Statnikov A (2003). "Algorithms for Large Scale Markov Blanket Discovery." In *Proceedings of the Sixteenth International Florida Artificial Intelligence Research Society Conference*, pp. 376–381. AAAI Press.

Tsamardinos I, Brown LE, Aliferis CF (2006). "The Max-Min Hill-Climbing Bayesian Network Structure Learning Algorithm." *Machine Learning*, **65**(1), 31–78.

Verma TS, Pearl J (1991). "Equivalence and Synthesis of Causal Models." *Uncertainty in Artificial Intelligence*, **6**, 255–268.

Whittaker J (1990). *Graphical Models in Applied Multivariate Statistics.* John Wiley & Sons, Hoboken.

Witten IH, Frank E (2005). *Data Mining: Practical Machine Learning Tools and Techniques.* 2nd edition. Morgan Kaufmann.

Yaramakala S, Margaritis D (2005). "Speculative Markov Blanket Discovery for Optimal Feature Selection." In *ICDM '05: Proceedings of the Fifth IEEE International Conference on Data Mining*, pp. 809–812. IEEE Computer Society, Washington, DC.

Affiliation:

Marco Scutari
Department of Statistical Sciences
University of Padova
Via Cesare Battisti 241, 35121 Padova, Italy
E-mail: marco.scutari@stat.unipd.it